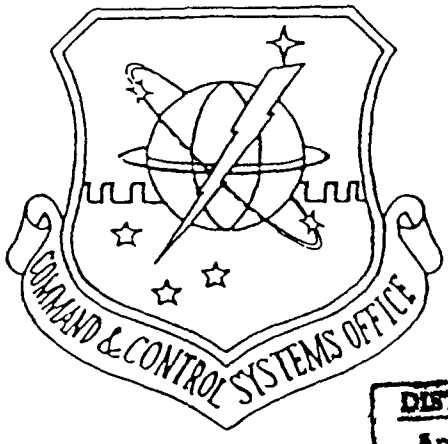
ADA* EVALUATION PROJECT

TRANSPORTABILITY ISSUES FOR ADA* SOFTWARE DEVELOPMENT

Prepared for

HEADQUARTERS UNITED STATES AIR FORCE
Assistant Chief of Staff of Systems for Command, Control,
Communications, and Computers
Technology & Security Division



Prepared by
Standard Automated Remote to AUTODIN Host (SARAH) Branch
COMMAND AND CONTROL SYSTEMS OFFICE (CCSO)
Tinker Air Force Base
Oklahoma City, OK 73145-6340
COMMERCIAL (405) 734-2457/5152
AUTOVON 884-2457/5152

* Ada is a registered trademark of the U.S. Government
(Ada Joint Program Office)

19 March 1987

TABLE OF CONTENTS

L I S T O F F I G U R E S

THIS REPORT IS THE SIXTH OF A SERIES WHICH DOCUMENT THE LESSONS LEARNED IN THE USE OF ADA IN A COMMUNICATIONS ENVIRONMENT.

ABSTRACT

This paper discusses Ada transportability. The first section of the paper provides some background information on the Ada evaluation task and defines the scope of the paper.

The second section of the paper looks at some of the main issues associated with transportability. The benefits of producing transportable software are covered along with some of the hidden costs and problems. The manager's role in developing transportable software and the effect of using the Ada language are discussed.

The third section looks at design issues. This section argues that if transportability is not defined as a requirement early in the development process, then the resulting software will not be very transportable. Design methods and software engineering principles that can aid in the development of transportable software are covered.

Section four addresses language issues. The need for well defined coding standards and programmer training is highlighted. Language issues that pose the greatest threat to transportability are covered in detail and references are provided to give the reader a more in-depth coverage of the subject.

The fifth section discusses some of the experiences that the Standard Automated Remote to Automatic Digital Network (AUTODIN) Host (SARAH) team encountered in developing transportable Ada software. The use of a logical kernel approach, standard pre-defined packages, and information hiding are discussed.

The final section summarizes some of the main points and provides some recommendations on the development of transportable Ada software.

Ada Evaluation Report Series by CCSO

Ada Training	March 13, 1986
Design Issues	May 21, 1986
Security	May 23, 1986
Micro Compilers	December 9, 1986
Ada Environments	December 9, 1986
Transportability	March 19, 1987
Runtime Execution	March 27, 1987
Modifiability	Winter 86-87
Project Management	Spring 87
Module Reuse	Fall 87
Testing	Fall 87
Summary	Fall 87

1. INTRODUCTION

1.1. THE ADA EVALUATION TASK

This paper is one in a series which seeks to help potential Ada developers gain practical insight into what is required to successfully develop Ada software. With this goal in mind, Air Staff tasked the Command and Control Systems Office (CCSO) to evaluate the Ada language while developing real-time communications software. The task involves writing papers on various aspects of Ada development such as training, Ada design, environments and security issues. This paper discusses transportability issues.

CCSO chose the Standard Automated Remote to AUTODIN (Automatic Digital Network) Host (SARAH)[1] project as the vehicle for the Ada evaluation. SARAH is a small to medium size project (approx. 40,000 lines of executable source code) which will function as a standard intelligent terminal for AUTODIN users and will be used to help eliminate punched cards and paper tape as a transmit/receive medium. The development environment for SARAH consists of a number of IBM PC AT, Zenith Z248, and Zenith Z150 microcomputers. Source code is compiled on the PC ATs and Z248s using Alsys Ada compilers and the object code can be targeted to any of the development microcomputers configurations. The SARAH software will run on a range of PC XT, PC AT, and compatible microcomputers under the MSDOS operating system (version 2.0 or higher).

1.2. BACKGROUND

The spiraling cost of software, the ever increasing need for larger and more complex software systems, and rapid advances in computer hardware technology have motivated planners to look closely at the question of software transportability. Traditionally, software applications have been rewritten or extensively modified if the application needed to be transported to another machine. This approach has been costly and has played a major part in what is often termed the 'software crisis'[7]. Faced with the problem of spiraling software costs and a projected lack of trained software professionals, the U.S. Department of Defense initiated the Ada program. Ada provides a good set of tools for producing transportable software but there are many other issues that must be considered if transportability goals are to be realized.

Transporting entire applications to other target environments can provide large cost benefits but there are also costs associated with producing transportable software. Software that has been designed to be transportable could be slower, larger in size, and could take longer to produce. Planners need to consider these hidden costs while determining the transportability requirements.

After the requirements have been established, design and coding standards should be established and adhered to by all members of the development team. Transportability must be considered throughout the development cycle or the software will not exhibit the degree of transportability specified in the requirements.

One of the design and development goals for the SARAH project is transportability. The SARAH software is targeted to a range of 'off the shelf' microcomputers. Because of the large cost of producing software and falling hardware costs, the SARAH planners foresaw the need to build a system that could easily be transported to new microcomputer architectures when they became available.

1.3. PURPOSE

The purpose of this paper is to:

- Provide details on the design and development issues associated with producing transportable Ada software.
- Outline some of the effects that transportable software may have on runtime execution, size, and the development times.
- Highlight some of the experiences and lessons learned during the design and development of transportable SARAH software.

1.4. SCOPE AND CONSTRAINTS

The terms transportability and software reuse are often used interchangeably. This paper discusses transportability of Ada software; but what does the term transportability actually mean and how does it differ from reusability? A good definition of software transportability is: "the capability of an application to be used again in a different target environment than the one for which it was originally developed"[5]. This is different from the definition of reusability which implies that components of the application are used in some new application. Indeed, software could be very portable but the components of the software may not be reusable because of tight coupling between the modules. Issues associated with reusability will be covered in a subsequent paper.

Since the SARAH software is currently still under development, the total effects of transportability cannot be fully determined. Software developed for SARAH has only been targeted to IBM PC AT, IBM PC XT, Zenith Z150, Zenith Z248, and compatible microcomputers. During initial research and experimentation,

some experience was gained in transporting Ada code between a Digital Equipment Corporation VAX 11/780, a Burroughs XE 550 Megaframe, a Zenith Z150 microcomputer, and an IBM PC AT.

2. TRANSPORTABILITY ISSUES

This section of the paper addresses several important issues associated with transportability. The benefits of developing transportable software, the effects of using the Ada language, manager education, and some of the costs associated with transportability are discussed.

2.1. THE BENEFITS

Overall software costs can be significantly reduced by producing transportable software. If complete applications can be moved to different target environments with very little modification, then significant cost benefits can be achieved. The Ada Europe guidelines on transportability[2] provide a formula for assessing transportability:

$$T = 1 - \frac{\text{cost of re-implementation on new target}}{\text{cost of original implementation}}$$

The value of T must be a positive fraction if any benefits are to be gained by moving an application to another target environment. If transportability is a development goal, then designers should aim at producing a system that will achieve a high portability fraction for the life of the software.

Implementors should aim at developing systems where the life of the software is longer than that of the target machine. Software applications are becoming larger and more complex and so the costs associated with developing such software are increasing dramatically. At the same time, hardware costs are falling and hardware performance is improving. Since software is one of the major costs of a system, planners are concerned about protecting the software investment. One way to achieve this is through transportability. When hardware becomes unmaintainable or a performance upgrade is required, the software should allow for movement to the new target hardware with little change.

2.2. ADA'S ROLE

The rising costs of software development and maintenance was a driving force for establishing the Ada program. Many of the language features and language management practices were defined to aid software transportability. For example, Ada provides support for concurrent operation directly with the language. Multi-tasking has traditionally been the domain of the assembly programmer who used special facilities in the operating system or runtime executive to provide concurrency. Since real-time tasking applications were very dependent on the underlying hardware and operating system, they were not very transportable. The Ada tasking model allows programmers to develop multi-tasking

applications without having to resort to using machine dependent code. As will be discussed later in this paper, the development of transportable multi-tasking applications in Ada is not without its problems but Ada provides the facilities for dealing with many of the problems associated with developing transportable multi-tasking applications.

A major obstacle to transportability has been the large number of languages in use and the number of subsets and supersets of these languages. The compiler packages were generally provided by the hardware vendors who would provide additional features to help market their products. The result was that even though an application was coded in, say, COBOL, the code could not be compiled on another implementation because the version of COBOL on the new machine (lets say COBOL+++) had different features from the COBOL on the existing machine. Ada has an advantage over these older languages because Ada is a strictly controlled standard and so there can be no subsets or supersets of the language. The U.S. Department of Defense has enforced language control through its Ada validation procedures and by trademarking the Ada name.

The Ada language alone will not yield transportable software[4]. The language is simply a vehicle for implementing the design. If software is to be transportable, then appropriate planning needs to be done early in the project and the transportability goals must be considered throughout the development cycle. Transportability just 'won't happen' simply because the Ada language is used for development.

2.3. THE COSTS OF TRANSPORTABILITY

The development of transportable software is not without its costs. Some of the costs that must be considered are:

- Size of resulting code,
- Speed of execution, and
- Development time.

In general, a software application that has been designed to be transportable will be larger. For example, if an application is to work with a number of different operating systems, a layer of abstraction will be needed to hide the operating system features from the modules in the application. In SARAH, the operating system filenaming conventions are hidden in a package called Disk_Defs. The rest of the application knows nothing of the filenaming conventions (i.e. how many characters are allowed for the disk drive name, the characters used as delimiters etc.); Disk_Defs does the conversion from the filename record structure used internally and the conventions used by the operating system. The benefit of this approach is that if a different operating system is used, then only Disk_Defs needs changing. However,

this added layer of abstraction will add to the amount of code required and hence the overall size of the application. The extent of the size increase will, to a large degree, depend on the skill of the designers and programmers.

Execution speed may not be as fast if the software is designed to be highly transportable. Working through an additional layer of abstraction will ultimately slow down the application. To illustrate, consider the filenaming example once again. A simple string type could have been used as the data structure for filename objects. The structure of this type would be globally visible to all SARAH modules. Each module in the application could then perform string manipulation to change the filename parameters (e.g., the drive name, directory path name, etc.). There are several other security and engineering implications that must be considered with this approach but there is little doubt that the approach would be faster than using a controlled type which has its own set of tools to perform conversions.

Design and development time will be longer if transportability is a major goal for development. Designers and programmers need to be trained to develop transportable software. There are many aspects of the Ada language and specific design techniques that must be understood if the resulting software is to be transportable. This additional training takes time. In addition to training for transportability, design and coding time could be longer. More time will generally be required during design to consider transportability criteria. Also, the time taken to code the design will be longer because of the additional restrictions that are put on the programmer.

2.4. A HIDDEN TRAP

Use of vendor supplied predefined packages can reduce potential transportability. Vendor supplied predefined packages are those delivered with the compiler and for which there is no available source code. They are analogous to the standard Ada predefined packages such as Text_IO. Several companies now provide their own predefined packages. For example, Alsys Inc. of Waltham MA provides two such packages. One package provides low level Disk Operating System (DOS) functions and the other provides bit manipulation facilities.

In many applications these low level features are necessary because the standard Ada packages such as Text_IO don't provide all the required features. For example, if the contents of a disk directory are required, then the programmer must resort either to writing a low level procedure for doing this or use a vendor supplied package.

A major danger is that if these packages are used for development, then the facilities provided by the vendor packages have to be provided by the developer (or the new compiler implementation) if the application is to be compiled using

another compiler. Since these packages are non-standard, the interfacing specifications could be different and quite a lot of work may need to be done to move the application. Developers need to be extremely careful in the use of these vendor supplied predefined packages or the product will be difficult to transport to different compiler implementations. The SARAH designers realized the dangers of these packages and their use has been restricted to areas where the features are absolutely necessary.

The problem may even be worse in the future because many vendors are beginning to provide other more complex packages. For example, Alsys has indicated that a Math package will be available in the next release of the compiler. Other companies such as Meridian are planning to provide support for everything from light pens to dynamic string handling. To avert this potentially devestating transportability problem, secondary standards for the more common packages such as string handling, mathematics, and data bases need to be quickly defined. If this is not done, there will be a proliferation of vendor supplied packages and transportability will suffer.

2.5. MANAGEMENT TRAINING

Manager education is important if transportability goals are to be realized. Management must understand that if transportability is identified as a development requirement, then there will be some trade-offs in terms of size, cost, and execution speed. Moreover, the manager must understand that transportability must be considered throughout the development cycle; transportability is not something that can be added at the end of the project. One of the current myths about Ada is that its use will ensure product portability. Managers should be shown that Ada is no better than any other language unless the correct procedures and practices are applied during design and development. Considering the possible pitfalls, transportability is seen as an important topic and should most definitely be covered in Ada manager training courses.

3. DESIGN FOR TRANSPORTABILITY

Design plays a major part in whether or not software will be transportable. Some of the factors that must be considered are: the need for transportability, the design methods that support transportability, and the modern software engineering practices that must be applied if transportability goals are to be achieved.

3.1. TRANSPORTABILITY REQUIREMENTS

If transportability is an important criteria for the software being developed, then it should be established as one of the system requirements. This should be done early in the project and tracked throughout the development cycle. A major question that should be answered is: What degree of transportability are we looking for and what payoffs do we expect?

The degree of transportability needs to be established after considering the trade-offs. A requirement that specifies that the software will be transportable is meaningless. One hundred percent transportability is almost impossible to achieve. Do we want the software to be transportable to a certain class of machine or should the software function independent of operating system characteristics? Is transportability even a criteria that we should be looking at? There are many cases where transportability is not particularly important and so the costs of making the software transportable may outweigh the advantages. Planners and developers need to properly define the transportability criteria required for a project so that a definite direction can be established early in the development cycle.

Transportability is a SARAH requirement. The main transportability requirement for SARAH is that the software should be easily transportable to other microcomputers as they become available. Since the advances in microcomputer technology are rapid and the cost of 'off the shelf' microcomputers is steadily falling, the life of the SARAH software needs to be longer than the life of the hardware. Indeed, since the project started, additional targets have been identified. Initially, the software was to only be targeted to the Zenith Z150, but since then there has been a requirement to add the Z248, Z200, and the Sperry PC to the list. No doubt, the release of the new Z386 which uses the advanced 80386 microprocessor will also end up being one of the target machines. In addition to different microcomputer hardware, the SARAH planners saw the need to make the application operating system independent. Although the MSDOS operating system is currently a defacto standard for microcomputers, during the life of the software there may be a need to host the software on a machine that uses a different operating system (e.g. UNIX).

3.2. DESIGN METHODS

An object oriented approach[7] to design can help identify and group objects and procedures that will not be transportable. One of the major benefits of this approach is that the resulting Ada code maps well to the design. For example, an object oriented approach may allow the designer to identify the printer control codes as objects which would not be transportable between different systems. These objects and the functions that manipulate the objects could be put into a separate package so that if the new system has a printer that uses different control codes, then only the one package needs changing. The programmer involved in transporting the software to another system is spared the frustration of sorting through all the code to find the parameters that need to be changed for the new printer. Ada packages and object oriented techniques therefore play a big part in encapsulating and isolating machine dependencies. In a large system, these packages may be logically collected into a subsystem which contains all the machine dependent code. This approach will be described in more detail later in this paper.

Although an object oriented approach can be helpful in design, this does not mean that an object oriented approach is the only methodology that should be used to design Ada software. Certainly other design paradigms and methodologies need to be considered in the total design approach[10].

3.3. INFORMATION HIDING AND ABSTRACTION

Information hiding and abstraction are important software engineering principles that can aid in producing transportable Ada software. Ada has various language features and constructs that support these principles. For example, packages can allow for implementation independent specifications and can help localize and hide machine dependencies.

Abstract interfaces are important for transportability. To illustrate this point, consider a driver for a mouse device. Several functions and procedures are needed by user programs to interface to the mouse. For example, the user programs need functions and procedures to provide information on which button was pressed and to provide grid co-ordinates. The details of how this is implemented should be hidden from the user so that if the software is transported to another system with a different mouse device, only the implementation details within the mouse package need change; the interface to the rest of the system remains intact. The modules that use the mouse package interface with the package specification which does not change.

4. LANGUAGE ISSUES

Language issues related to transportability are covered in detail in several publications[2 3 5]. This section will cover some of the issues that the SARAH team has identified as critical for transportability.

4.1. CODING STANDARDS

Specific coding standards need to be established and used throughout development if the resulting software is to be transportable. Planners need to look at the published guides on transportability[2,3] and establish the appropriate design and coding standards in the project documentation. Management should then ensure that programmers comply with the coding standards. The SARAH project used DOD-STD 2167 as the documentation standard and the design and coding standards for SARAH were defined in the SARAH Software Standards and Procedures Manual (SSPM).

Programmers must be educated in using a coding style that is consistent with the transportability objectives. Most basic Ada training courses discuss the language features but do not give specific instruction on transportability objectives. There are many language issues that need to be addressed and programmers cannot be expected to produce portable code if they are not made aware of these issues. Once the coding standards have been defined, the programmers should be instructed on the specific transportability objectives of the project, the techniques that must be employed (as defined in the standards manual), and why the techniques are important if the application is to be transportable.

4.2. DYNAMIC ALLOCATION

Dynamic allocation is one area where portability problems can arise so care needs to be taken to identify the potential problems. One of the main problems is that there is uncertainty as to how much storage is taken by dynamically allocated objects. As such, an application that runs on one machine may raise the Storage_Error exception on another implementation.

Another problem is that different compiler implementations deallocate objects at different times. The Ada Language Reference Manual (LRM)[6] does not specify when deallocation should take place. As such, software that dynamically deallocates objects may not execute in the same way under different implementations. This could cause major problems for transporting real time software. This problem can be overcome to a large extent by the use of a 'free list' approach[5].

4.3. DISCRETE TYPES

Potential portability problems can arise because discrete types have a range that is machine dependent and because Ada does not dictate that range checks must be done on subexpressions.

If an application is to be transportable, then type names Short_Integer and Long_Integer should not be used explicitly. Rather, range constraints should be used to define integers. To illustrate, consider the following example. Software is produced on a 32 bit machine which will support an integer range of over four billion. The programmer uses the following type declaration:

```
subtype Count_Type is  Long_Integer;
```

Now, the application needs to be transported to an 16 bit machine which has an integer range of -32767 to 32767. On this implementation Long_Integer is not defined. Before the software will run on the new machine, all the declarations that used Long_Integer will have to be changed. Also, this is a poor programming practice because, in general, the whole integer range is not required. Rather than define the explicit integer type, the programmer could have defined:

```
type Count_Type is range 1..50;
```

The range of Count_Type is now restricted, but more importantly, the declaration is now transportable. For the 16 bit machine, the base integer type is Integer and so Count_Type will be defined as a sixteen bit integer. The 32 bit machine had Long_Integer defined as its base type and so Count_Type will be defined as a 32 bit integer. Code changes are therefore not required when the application is moved to the new machine.

Another potential transportability problem arises because Ada does not call for range checks on subexpressions. A subexpression could therefore have a value outside that specified for its type. To illustrate this point with an example: we again develop our application on a 16 bit machine and use:

```
type Count_Type is range 1..50;

My_Count : Count_Type := 12;
```

and then use the expression:

```
My_Count := My_Count**2 - My_Count*10;
```

The result of the calculation (24) is within the range of Count_Type and the program runs fine on the 16 bit machine. When the program is transported to an 8 bit machine, Numeric_Error is raised at run time. What happened? Because Ada does not call for range checks on subexpressions, the base integer type is used. In the case of the 16 bit machine, the base type Integer was used and so, when the subexpression My_Count**2 was

evaluated, the result was 144 and this was within range. But, for the eight bit machine, Short_Integer was the base type and so 144 was out of range and Numeric_Error was raised. As such, we created a runtime problem when we transported the application to an 8 bit machine. This situation can be potentially dangerous, because it renders the software unreliable.

If the programmer was aware of the transportability problems that could arise, he may have factored the expression:

```
My_Count := My_Count*(My_Count - 10);
```

Now the expression would work equally well on both implementations.

Although the previous example discussed eight and 16 bit implementations, the same type of problem can occur when transporting Ada software between any machines with any different size data words. Similar problems can also arise when using floating point or fixed point numbers. The main aim of this section has not been to outline all the transportability problems that could arise through Ada types, but rather to show there are potential problems. More in-depth coverage of these problems can be obtained from several other publications[2 3 5].

4.4. TASKING

The development of transportable real-time software that contains tasking can be very difficult. Concurrent processing has traditionally been seen as the arena of assembly programmers who specialize in interfacing applications to operating systems or runtime executives. Ada provides support for concurrency within the language but, even so, issues associated with multi-tasking in real time systems pose major obstacles to transportability.

A major problem is that execution speeds vary from machine to machine. If software depends on execution speed, the transported application may behave differently on the new target. When using Ada, there are several issues that must be carefully considered. For example, differences in scheduling algorithms, the different lengths of task queues, and the order of task activation all provide potential transportability problems.

Ada compilers need not implement the task scheduling algorithm in the same way. The Ada Language Reference Manual(LRM)[6] does not specify which type of task scheduling should be implemented. For example, not all of the current compilers use a time slicing algorithm. In the case of the Alsys Version 1.3 compiler, task switching occurs only at task synchronization points[11]. If an application was developed assuming that a time slice algorithm would be used and this application was moved to an implementation where task switching occurs only at synchronization points, then there is a high probability that one of the tasks would take

control of the processor and not allow other tasks to execute. The code would have to be modified to insert synchronization points at strategic locations (possibly by adding delay statements) so that the tasks would switch and so simulate the action of time slicing.

The length of task queues can vary between different compilers. Ada does not specify the length of these queues. If the number of elements that can be queued up to an entry point is less on the new target system, there is a possibility that some events could occur in a different order than that originally planned for. This can be a very serious problem because the application could become unreliable after it is transported to the new target. As such, designers and programmers should not make any assumptions on how big the task queues will be on a particular implementation. If a large number of elements are to be queued, then for the sake of transportability, an independent queuing system should be used.

No assumption should be made about the order of task activation. The LRM deliberately leaves this issue undefined. The Ada Europe Guidelines[2] indicate that making such an assumption could "..lead to a runtime surprise". To ensure transportability for applications which rely on a specific order of task activation, the activation and execution sequence should be performed by the application. The same could also be said for task termination. In the SARAH system, the application itself controls when tasks will execute and when they will be terminated. This not only aids in providing a more transportable application but it also reduces potential testing and maintenance problems.

4.5. ELABORATION ORDER

All library units and unit bodies must be elaborated before a main program can be executed; but Ada does specify the order of elaboration for unit bodies. Even so, the elaboration order may vary with different implementations.

Consider the example in Fig 4-1. Printer_Params contains some implementation specific printer constants. Printer_Driver gets these implementation specific values from the Printer_Params package. The LRM specifies that Printer_Params must be elaborated before Printer_Driver. Also, the body of Printer_Params must be elaborated after its specification. However, an implementation is free to elaborate the body of Printer_Params before or after the elaboration of Printer_Driver. If the body of Printer_Driver is elaborated before the body of Printer_Params, then Characters_Per_Line will not be initialized. By reversing the elaboration order, Characters_Per_Line will be correctly initialized to 80. If Ada software is to be reliable after it is transported, designers and programmers must look closely at possible elaboration problems during development.

```
-------------------------------------------------------------
package Printer_Params is

   type Length_Type is range 1..132;

   Line_Length : Length_Type;

end Printer_Params;


package body Printer_Params is

   begin

      Line_Length := 80;

end Printer_Params;
-------------------------------------------------------------
```

```
-------------------------------------------------------------
with Printer_Params;

package Printer_Driver is

   Characters_Per_Line : Printer_Params.Length_Type
                       := Printer_Params.Line_Length;

end Printer_Driver;
-------------------------------------------------------------
```

Fig. 4-1: Example of Possible Elaboration Problems

4.6. PRAGMAS

Every effort should be made not to use implementation defined pragmas. Pragmas are directives to the compiler and are characterized as language defined and implementation defined. Implementation defined pragmas may have different meanings for different implementations. If these pragmas are used, their use should be well documented and the documentation should highlight the parts of the code where the pragmas are used. Preferably, the pragma should be encapsulated in one of the packages which contain machine dependencies.

5. TRANSPORTABILITY EXPERIENCES WITH SARAH

As discussed earlier, transportability requirements for SARAH were established early in the development cycle. This section describes the approach used and highlights some of the main transportability features of the SARAH system.

5.1. THE LOGICAL KERNEL

Early in the design phase, the SARAH designers identified a logical kernel to aid in transportability. As shown in Fig 5-1, the kernel shields the application from the machine and DOS dependencies. When transporting the application to another target environment, only the code in the kernel packages needs to be changed.

The kernel consists of a logical grouping of packages. For example, the machine dependent code for the Print_Manager subsystem is encapsulated in a package called Printer_Kernel. This package is a part of the Print_Manager subsystem, but it is also identified as a part of the logical SARAH Kernel. If the Print_Manager subsystem is to be reused, then the entire subsystem can be removed along with its kernel package. The programmer would then need to modify the Printer_Kernel package for the new target. If the entire SARAH application is to be reused (or transported), then the programmer needs to look at all the packages that are defined in the SARAH Kernel, one of which would be the Printer_Kernel package.

5.2. INFORMATION HIDING

Information hiding was used extensively in the design of SARAH so that specific low level information could be hidden from the majority of the software elements. As discussed earlier, the use of information hiding and abstraction are important software engineering principles that need to be applied when developing transportable software.

SARAH provides several good examples of how information hiding and abstraction have been used to promote transportability. For example, the SARAH system is not tied to the filenaming conventions of the operating system. The SARAH subsystems that use disk input/output operate with a Filespec_Type. The structure of this type is invisible to all of the subsystems except Disk_Manager. Even within Disk_Manager, the only package that knows anything of the DOS filename structure is Disk_Defs (which incidentally is a SARAH Kernel package). As such, redundant information is hidden from those elements that do not specifically need the information. The result is that the application is more easily transported because the implementation dependent information has been localized.

The Visual Display Terminal Manager (VDT_Manager) subsystem also makes good use of information hiding to promote transportability. To implement a fast windowing scheme, VDT_Manager resorted to using direct screen addressing. However, the details of how this is implemented is hidden in a kernel package. The SARAH subsystems that need to write to the screen use abstract interfaces that show no relationship to the physical properties of the display. Functions and procedures such as Open_Transient_Window and Display_Help_File are provided for the subsystems that interface to the VDT_Manager. By hiding the implementation details and providing abstract package interfaces, the VDT_Manager designers were able to enhance the potential transportability of the SARAH system.

5.3. STANDARD PRE-DEFINED PACKAGES

The SARAH developers attempted to use standard pre-defined packages wherever possible. These packages are well defined and standardized as part of the Ada language. As such, they must be provided with all validated compilers and so aid in producing transportable software. As discussed earlier, some vendors supply their own predefined packages. The SARAH team was careful not to use these packages unless there were significant benefits to be gained. For example, since the standard input/output packages do not provide facilities for reading disk directories, these facilities needed to be supplied by some other means. After a careful look at the trade-offs involved in developing the routines 'in-house' or using the Alsys DOS package, the team members chose to use the DOS package. The Alsys DOS package was seen as a part of the logical kernel and in each case where routines were used from this package, the team carefully examined all possible options.

Although there were definite transportability benefits to using the standard Ada packages, the SARAH team was concerned about their effect on execution speed and code size.

Care needs to be taken when considering using the standard predefined packages if execution speed is critical. Since the SARAH system is user oriented, there is a large amount of interaction between the user and the system. The display features a number of windows and an arrangement of bar and pull-down menus. Initial experiments with the standard TEXT_IO package showed that the routines provided could not be used to efficiently implement the user interface. Using TEXT_IO would have used an unacceptable amount of the total available computer power. A separate subsystem, called VDT_Manager, was developed to provide the user interface.

The size of the resulting code is also an issue when using the standard pre-defined packages. For example, the Alsys Version 1.3 compiler adds some 26,000 bytes to the size of the executable file if TEXT_IO is used in the application. If only one or two routines are required, then this is a high price to

pay, particularly if the amount of memory is limited (as it is with the Z150 SARAH target). For this reason, SARAH does not use TEXT_IO. In the future, compilers may employ 'smart' linkers and so only the required code will be added to the executable file. Until then, care needs to be taken when considering the standard pre-defined packages if code size is an important consideration. This is one of the transportability trade-offs that needs to be made.

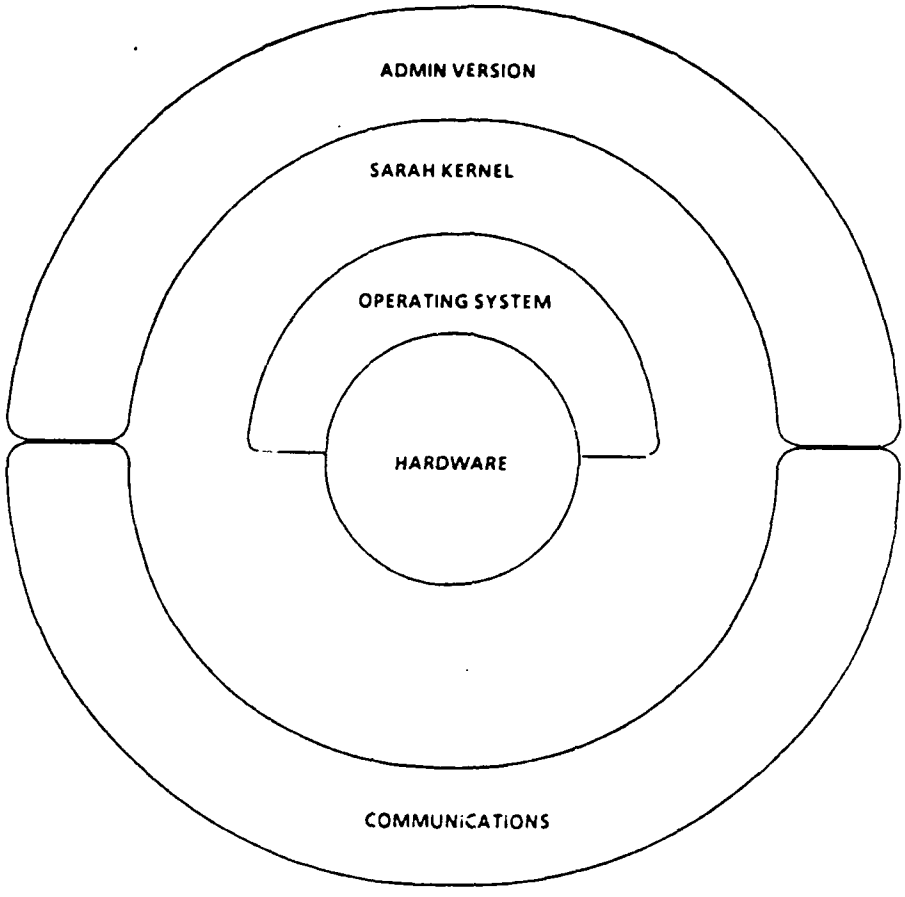


Fig. 5-1: SARAH Logical Kernel

6. SUMMARY AND RECOMMENDATIONS

6.1. SUMMARY

Transportability can dramatically reduce overall software costs. If entire software applications can be reused without significant modification, then enormous cost benefits can be achieved. Indeed, as our systems become larger and hardware costs continue to fall, economic pressures will dictate that the software must be easily transportable to new target systems. Requirements will indicate that the software must have a longer life than the hardware.

Transportability must be addressed early in the project. To be sure, transportability is not something that can be built in after the software is developed. Planners need to establish the transportability requirements for a project and the objectives need to be clearly defined. Once the objectives are defined, specific design and coding standards need to be provided for the project. These standards must then be adhered to throughout the development cycle.

Managers must be aware that although the Ada language will support the development of transportable software, Ada's use will not guarantee transportability. Ada must be viewed as a tool for developing transportable software. Effective design techniques, well defined goals, and the skill of the software developers will all have a major bearing on transportability. There are many language and design issues that must be considered when developing transportable Ada software.

Transportability has its hidden costs. Some of the costs associated with producing transportable software are increased code size, slower execution speeds, and increased development costs. Planners and managers must understand that there will be trade-offs to be made, and that these trade-offs will be closely coupled to the portability objectives. The degree that these hidden costs will affect the project will to a large extent depend on the experience and skill of the designers and programmers on the development team.

6.2. RECOMMENDATIONS

Recommendations are:

- Avoid using non-standard pre-defined packages wherever possible.
- If transportability is a production goal, then transportability issues must be addressed early in the development cycle.
- Managers, designers, and programmers must be educated in transportability issues.
- The Ada community should quickly establish secondary language standards to prevent the proliferation of vendor predefined packages.
- Transportability criteria should be closely studied and considered when establishing of the coding and design standards for a project.

A. REFERENCES

[1] "SARAH Operational Concept Document", Command and Control Systems Office, US Air Force, 5 September 1986.

[2] WALLIS P.J.L., WICHMAN B.A., NISSEN J.C.D, et al, "Ada Europe Guidelines for the Portability of Ada Programs", ACM Ada Letters Vol1 No 3 (March-April 1982) pp 44-61.

[3] "Ada Portability Guidelines", National Technical Information Service, No. AD A160 390, March 1985.

[4] HOWE R.G., HAZLE M. et al, "Program Managers Guide to Ada", USAF, ESD-TR-85-159, May 1985.

[5] AUSNIT C., BRAUN C., et al, "Ada Reusability Guidelines", National Technical Information Service, No. AD A161 259, April 1985.

[6] U.S. Department of Defense, "Reference Manual for the Ada Programming Language", ANSI/MIL-STD 1815A, Jan 1983.

[7] BOOCH G., "Software Engineering with Ada", Benjamin/Cummings, Menlo Park, California, 1983.

[8] NISSEN J., WALLIS P., "Portability and Style in Ada", Cambridge University Press, 1984.

[9] "Defense System Software Development", DOD-STD-2167, Department of Defence, Washington D.C. 20301.

[10] "An Architectural Approach to Developing Ada Software Systems", Command and Control Systems Office, Tinker Air Force Base, Oklahoma, 21 May 1986.

[11] BROGSOL B., AVAKIAN A.S., GART M.B., "Alsys Ada Compiler for tne IBM PC", Proceedings of First International Conference on Ada Language Applications for the NASA Space Station, June 1986.

[12] "Usage and Selection of Ada Microcomputer Compilers", Command and Control Systems Office, Tinker Air Force Base, Oklahoma, 9 December 1986.